

REMARKS

This application has been carefully considered in connection with the Examiner's Office Action dated December 23, 2008. Reconsideration and allowance are respectfully requested in view of the following.

Summary of Rejections

Claims 1-37 were pending at the time of the Final Office Action.

Claim Rejections – 35 U.S.C. § 101

Claims 32-37 stand rejected under 35 U.S.C. § 101 because the claimed invention is allegedly directed to non-statutory subject matter.

Claim Rejections – 35 U.S.C. § 103

Claims 1-10, 12, 16, 18-20, and 31-37 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty, et al., U.S. Patent No. 7,047,525 (hereinafter "Prunty") in view of Chan, et al., U. S. Patent No. 6,836,889 (hereinafter "Chan").

Claims 11 and 14 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan as applied to claim 1, and further in view of Generous, et al., U.S. Publication No. 2002/0120697 (hereinafter "Generous").

Claims 21 and 22 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan as applied to claim 12 and further in view of Seidman, et al., U.S. Publication No. 2002/0199032 (hereinafter "Seidman").

Claim 23 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan and further in view of Seidman as applied to claim 22, and further in view of Smith, et al., U.S. Publication No. 2003/0167355 (hereinafter "Smith").

Claims 24, and 27-30 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan and further in view of Seidman, and further in view of Smith as applied to claim 23 and further in view of Haynie, et al., U.S. Publication No. 2006/0036448 (hereinafter "Haynie").

Claims 25 and 26 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan and further in view of Seidman and further in view of Smith as applied to claim 23, and further in view of Whalen, et al., U.S. Patent No. 5,948,066 (hereinafter "Whalen").

Summary of Response

Claim 32 is amended herein.

Claims 1, 22, and 33-37 remain as previously presented.

Claims 2-12, 14, 16, 18-21 and 23-31 remain as originally submitted.

Claims 13, 15, and 17 were previously canceled.

Remarks and Arguments are provided below.

Summary of Claims Pending

Claims 1-12, 14, 16, and 18-37 are currently pending following this response.

Applicant Initiated Interview

Applicants thank Examiner Charles E. Anya for his time and consideration of the arguments presented in the telephone interview on March 4, 2009. In the interview, Examiner Anya further considered the applied art in view of the Applicants' arguments. Examiner Anya indicated that a further search may be performed. A detailed discussion of the differences between the applied art and the claim limitations follows.

Response to Rejections under Section 101

Claims 32-37 were rejected under 35 USC § 101 because the Office Action alleges that the claimed invention is directed to non-statutory subject matter.

The Office Action rejected claims 32-37 under 35 USSC § 101 as being directed to software *per se*. Claims 32-37 have been amended herein to recite the functional descriptive material or software limitations as being stored on a computer-readable storage medium. MPEP 2106.01(I) states, "[A] claimed computer-readable medium encoded with a computer program is a computer element which defines structural and functional interrelationships between the computer program and the rest of the computer which permit the computer program's functionality to be realized, and is thus statutory. See Lowry, 32 F.3d at 1583-84, 32 USPQ2d at 1035. ...When a computer program is recited in conjunction with a physical structure, such as a computer memory, USPTO personnel should treat the claim as a product claim." Applicants respectfully submit that the amendments to claims 32-37 now recite statutory subject matter. Accordingly, Applicants respectfully request the rejection under 35 USC § 101 be withdrawn.

Response to Rejections under Section 103

According to the United States Supreme Court in *Graham v. John Deere Co. of Kansas City*, an obviousness determination begins with a finding that **“the prior art as a whole in one form or another contains all” of the elements of the claimed invention.** See *Graham v. John Deere Co. of Kansas City*, 383 U.S. 1, 22 (U.S. 1966).

Claim 1:

In the Office Action dated December 23, 2008, claim 1 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan.

Independent claim 1 recites:

1. A method of accessing an Enterprise Java Bean (EJB) by a non-Java application within a computing environment, comprising:
 - a) **making a call**, by the non-Java application, **to a client library**, wherein the call includes input parameters, and wherein **the client library is a linkable library that is dynamically linked to the non-Java application**;
 - b) **invoking a function within the client library to construct an HTTP request** corresponding to the input parameters of the call made from the non-Java application, **wherein numeric primitive data types of the non-Java application in the calling input parameters are converted into a corresponding text representation in the HTTP request**;
 - c) passing the HTTP request from the client library to an EjbServlet;
 - d) invoking a method on an EJB by the EjbServlet based upon the HTTP request;
 - e) returning information from the invoked method from the EJB to the EjbServlet;
 - f) decoding the returned information into an HTTP response string by the EjbServlet;
 - g) **transmitting the HTTP response from the EjbServlet to the client library**; and
 - h) parsing and converting the HTTP response by the client library into return information compatible with the non-Java application and then passing the return information from the client library to the non-Java application, **wherein text-**

represented numeric values extracted from the HTTP response are converted into a corresponding numeric primitive data type of the non-Java application.

Applicants respectfully submit that the art of record does not establish a *prima facie* case of obviousness as to the pending claims because the art of record fails to teach or suggest all of the claim limitations. Specifically, Prunty in view of Chan fails to teach or suggest converting numeric primitive data types of a non-Java application in calling input parameters into a corresponding text representation in a HTTP request, converting text-represented numeric values extracted from a HTTP response into a corresponding numeric primitive data type of the non-Java application, or a client library that is a linkable library dynamically linked to a non-Java application.

I. Prunty in view of Chan does not teach or suggest a client library, where the client library is a linkable library that is dynamically linked to the non-Java application.

Claim 1 recites:

making a call, by the non-Java application, to a client library . . . wherein the client library is a linkable library that is dynamically linked to the non-Java application . . . invoking a function within the client library to construct an HTTP request . . . transmitting the HTTP response . . . to the client library . . . and then passing the return information from the client library to the non-Java application.

The Office Action concedes on page 5, "Prunty is silent with reference to the client library being a linkable library that is dynamically linked to the non-Java application." The Office Action relied on column 6, lines 54-67, and column 7, lines 1-5, of Chan as disclosing "the client library being a linkable library that is dynamically lined to the non-Java application." (Page 5) To provide further context to the cited section of Chan, column 6, line 42 - column 7, line 8 is reproduced below:

FIG. 3 is a simplified conceptual diagram of ...how the Access Bean 310 interacts with a client program 150 and a server-based software component, preferably in which the component is an enterprise bean 138 on an Enterprise JavaBeans server 130. Client program 150 in the context of the invention can be a Java program, such as an application, applet, or servlet or it can be another enterprise bean deployed in the same or different Enterprise JavaBeans server 130. ...Access Bean 310 is computer executable code that wraps around a software component, such as an enterprise bean, to facilitate access to and consumption of the reusable software component by one of many client programs 150. Access Bean 310 is preferably designed to support servlets and JavaServer Pages.TM. (JSP) programs ...but **can be used by any client program that needs to access a software component**, preferably an enterprise bean 138 on the server side. **Access Bean 310 may also be mapped to non-Java client environments**. Using Access Bean 310, consuming an enterprise bean is like consuming a Java bean because Access Bean 310 interacts with client program 150 through constructors 320 and 330 and methods as will be discussed. ...The enterprise bean 138 is on an Enterprise JavaBeans server 130 which preferably contains a plurality of enterprise beans. (bold added)

As described above and shown in Fig. 3 of Chan, an Access Bean 310 is executable code that wraps around a server side software component 138 to facilitate access to and consumption of the server side software component 138 by a client program 150. Also described above, the client program is preferably a Java program, the Access Bean 310 is a Java bean (see also Chan: column 2, lines 3-62 and column 7, lines 40-42), and the server side software component 138 is an EJB on an EJB server 130. Accordingly, Chan discloses that preferably, all of the elements in question are Java elements. This assertion is further supported by Chan in column 5, lines 13-17, which states, "In summary, in a preferred embodiment, this invention attempts to hide the complexity of programming to an enterprise bean from various Java client programs by providing a simple Java bean wrapper, called an Access Bean, for enterprise beans." In contrast, because the pending disclosure's client library is dynamically linked to a non-Java application, neither the client library nor the non-Java application are Java

components or execute in a Java environment.

As discussed in the telephone interview, the cited section of Chan also specifies, "Access Bean 310 may also be mapped to non-Java client environments." The Office Action has interpreted this solitary statement as a disclosure that client program 150 may be a non-Java client and may use the Java Access Bean 310 to access the EJB 138 on the EJB Server 130. Applicants submit that such an interpretation of this statement in Chan is inconsistent with the disclosure of Chan as a whole. A search of Chan did not locate any other references to non-Java environments. Applicants submit that the remainder of Chan's disclosure is directed to describing the client program 150, the Access Bean 310, and the EJB 138 as Java elements.

As noted above, Chan discloses in column 6, line 67 - column 7, line 2, "Access Bean 310 interacts with client program 150 through constructors 320 and 330 and methods as will be discussed." Chan goes on to discuss this interaction in column 7, lines 25-27 by stating, "[T]he client program 150 calls the necessary JavaBeans method 330 which in turn calls the corresponding business method on the enterprise bean 138." Accordingly, Chan discloses that the client program 150 interacts with the Access Bean 310 through JavaBeans methods. Applicants submit that the cited portions of Chan appear to be the only disclosure in Chan related to how the client program 150 interacts with the Access Bean 310. If the client program 150 is a non-Java client as asserted by the Office Action, how can a non-Java client program call JavaBeans methods? This is the precise problem to which the pending claims and disclosure are directed, namely the "problem of integrating Java components ...into non-Java environments" (Specification, paragraph 0005). Applicants submit that Chan is not concerned with and does not

provide a solution to such a problem.

In contrast with the interpretation presented by the Office Action and discussed in the telephone interview, Applicants submit a different interpretation of the disclosure of Chan in question should be maintained. In particular, Applicants submit that Chan's disclosure that "Access Bean 310 may also be mapped to non-Java client environments," should be interpreted as disclosure that the functionality of the Access Bean 310 may also be used in non-Java environments. Applicants submit that such an interpretation is consistent with the disclosure of Chan as a whole. The remainder of Chan is directed to the various elements being described as Java elements and the statement in question is simply an attempt by the drafter of Chan's disclosure to expand the scope of Chan's disclosure outside of just Java environments. This interpretation is also consistent with Chan's later expansive disclosure in column 14, lines 22-28, reproduced below:

While the preferred embodiment of this invention has been described in relation to the Java language, this invention need not be solely implemented using the Java language. It will be apparent to those skilled in the art that the invention may equally be implemented in other computer languages, such as object oriented languages like C++ and Smalltalk.

Therefore, Chan's disclosure when considered as a whole teaches a Java client program, an access bean, and an EJB server that all execute in a Java environment or a client program, an access component, and a server that all execute in a non-Java environment. Chan does not disclose mixing and matching components that execute in a Java environment with components that execute in a non-Java environment.

Furthermore, Chan's description of an Access Bean is not disclosure of a client library. Chan specifies the purpose of an Access Bean as "computer executable code that wraps around a software component, such as an enterprise bean, to facilitate access

to and consumption of the reusable software component by one of many client programs.” (Column 6, lines 54-58) Chan clarifies that “According to the principles of the invention, Access Bean 310 adapts an enterprise bean 138 to the JavaBeans programming model by wrapping or hiding the enterprise bean home interface 132 and remote interface 134 from the client program 150.” (Column 7, lines 17-20) Chan indicates that “In step 10 of Fig. 3, Access Bean 310 is instantiated thereby instantiating an enterprise bean 138 in the EJB server 130 with appropriate constructor 320 mapping to a particular home interface method 132.” (Column 7, lines 20-23) While Chan may provide disclosure of the client program 150 calling a JavaBeans method 330 on the Access Bean 310, Chan does not provide any disclosure or suggestion that the Access Bean is a client library as claimed. A search of Chan did not locate any references to a library. Even if Chan’s Access Bean was a client library, which it is not, Chan’s Access Bean is not a client library that is dynamically linked to a non-Java application. A search Chan locates only one reference to dynamic, in the context of creating dynamic web pages, and does not locate any references to link, links, linked, or linking.

II. Prunty in view of Chan does not teach or suggest converting numeric primitive data types in calling input parameters to corresponding text representations in HTTP requests or converting text-represented numeric values in HTTP requests to numeric primitive data types corresponding to non-Java applications.

Claim 1 recites, “wherein numeric primitive data types of the non-Java application in the calling input parameters are converted into a corresponding text representation in the HTTP request . . . wherein text-represented numeric values extracted from the HTTP

response are converted into a corresponding numeric primitive data type of the non-Java application.”

The Office Action relied on column 5, lines 34-39, of Prunty as disclosing “request is translated . . . (e.g., XML) (step 904).” (Page 4) Column 5, lines 34-51, of Prunty discloses:

FIG. 9 presents a flowchart illustrating an exemplary overview of the operation of an embodiment of the present invention. A request is made at a first computing platform (step 902). The request is translated into a predetermined format that is readable by both the first computing platform and the second computing platform (e.g., XML) (step 904). The request is then transmitted from the first computing platform to the second computing platform (step 906). At the second computing platform, the request is translated from the predetermined format into a format usable by the second computing platform (step 908). The second computing platform then processes the request and either acquires the requested data or performs the requested task (step 910). The result is then translated into the predetermined format (step 912), and transmitted to the first computing platform (step 914). The first computing platform translates the result from the predetermined format (step 916) and displays the result or otherwise makes use of the result (step 918).

Prunty discloses a request, which may have multiple input/output parameters, is translated into a predetermined format such as XML to build an XML request stream. However, Prunty does not teach or suggest converting numeric primitive data types in calling input parameters corresponding to non-Java applications to corresponding text representations in HTTP requests and converting text-represented numeric values in HTTP responses to numeric primitive data types corresponding to non-Java applications. Translation of parameters into an XML stream does not teach or suggest the specific conversion of numeric primitive data types in calling input parameters corresponding to non-Java applications to corresponding text representations in HTTP requests or the

specific conversion of text-represented numeric values in HTTP responses to numeric primitive data types corresponding to non-Java applications.

Accordingly, Prunty in view of Chan does not teach or suggest converting numeric primitive data types in calling input parameters to corresponding text representations in HTTP requests or converting text-represented numeric values in HTTP requests to numeric primitive data types corresponding to non-Java applications.

Claims Depending From Claim 1:

Claims 2-10, 12, 16, 18-20, and 31 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan.

Claims 11 and 14 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan in further view of Generous.

Claims 21 and 22 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan in further view of Seidman.

Claim 23 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan and Seidman in further view of Smith.

Claims 24 and 27-30 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan, Seidman, and Smith in further view of Haynie.

Claims 25 and 26 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan, Seidman, and Smith in further view of Whalen.

Dependent claims 2-12, 14, 16, and 18-31 depend directly or indirectly from independent claim 1 and incorporate all of the limitations thereof. Accordingly, for at least the reasons established in sections I and II above, Applicants respectfully submit that all

of the limitations of dependent claims 2-12, 14, 16, and 18-31 are not taught or suggested by Prunty in view of Chan and respectfully request allowance of this claim. Applicants respectfully submit that neither Generous, Seidman, Smith, Haynie, nor Whalen cure the deficiencies of Prunty in view of Chan.

Claim 22:

III. Prunty in view of Chan does not teach or suggest wherein buffers comprise an information text string and a format text string that comprises tags that comprise a name field, a start position field, and a length field used to locate data values in an information text string.

Claim 22 recites, “wherein the buffers comprise a data buffer and a format buffer, wherein the data buffer comprises an information text string comprising a text representation of all information returned from the invoked method, the format buffer comprises a format text string generated by the EjbServlet comprising format information about the information returned, and the format text string comprises tags that comprise a name field, a start position field, and a length field used to locate data values in the information text string.”

The Office Action relied on paragraphs 0024 and 0037 of Seidman to read on the limitations of “a start position field, and length field used to locate data values in the information text string.” (Page 11) Paragraph 0024 of Seidman discloses:

In accordance with this embodiment of the present invention, DeploymentCoordinator 7d receives a deferred response component deployment file utilizing any one of a number of methods that are well known to those of ordinary skill in the art. As will be described in detail below, the deployment file comprises one or more deferred response components along with a deployment descriptor text file (see FIG. 4) that

gives declarative instructions to EDRB Container 7b for each component. For example, and without limitation, DeploymentCoordinator 7d can poll a predetermined subdirectory of Enterprise DeferredResponseBean Container 7b for the presence of a new deployment file, or DeploymentCoordinator 7d can be invoked directly by way of an Enterprise JavaBean SessionBean that represents DeploymentCoordinator 7d in accordance with methods that are well known to those of ordinary skill in the art. Then, in response, DeploymentCoordinator 7d reads each of the components in the new deployment file, along with their associated deployment descriptors. In accordance with this embodiment of the present invention, the components read by DeploymentCoordinator 7d may either be a class or a serialized component instance. However, whenever the component read is a class, the class is instantiated in accordance with methods that are well known to those of ordinary skill in the art. Nevertheless, in either case, DeploymentCoordinator 7d deploys a component by making an entry in DeployedDeferredResponseBeanDictionary 7f. The component's deployment descriptor name is used as a key to DeferredResponseBeanPool 7h--but with N copies of the component instance as shown in FIG. 7k where N is the maximum number of threads designated in the deployment descriptor for the component. At the same time, DeploymentCoordinator 7d generates two objects implementing two client interfaces, and stores them in a directory service in accordance with any one of a number of methods that are well known to those of ordinary skill in the art. In accordance with one embodiment of the present invention, one of these client interface objects is stored at a name constructed by concatenation of "Requestor" with a name assigned to the component in the deployment descriptor, and the other one of these client interface objects is stored at a name constructed by concatenation of "Responder" with the name assigned to the component (see FIG. 6). In accordance with this embodiment of the present invention, Requesters are interfaces used by client systems (see FIG. 1) to interact with EDRB Container 7b, and Responders are interfaces that backend systems (FIG. 1) use to interact with EDRB Container 7b

Paragraph 0037 of Seidman references a deployment descriptor:

Whenever the method finally responds and returns an object as the return value, the deployment descriptor may have designated multiple output values for this method. If this is the case, the method element in the deployment descriptor will comprise one or more names of outputs. DeferredResponseBeanThread 7i extracts these outputs from the single return value of the method by calling "property read" methods on the single object. In accordance with this embodiment, the names of the property read methods are formed by concatenating a string "get"<output name> for each output name listed. A method with these names is then invoked on

the single return object for each output name listed. Then, the values obtained are assembled into a vector. These multiple outputs may then be returned to the client system utilizing the `returnOutputs()` method that returns a Vector.

Seidman discloses a method returning an object as a return value, and a deferred response bean thread using output names in a deployment descriptor to call “property read” methods on the single object for each output name listed. However, Seidman does not teach or suggest tags for a name field, a start position field, and a length field used to locate data values in an information text string. Because Seidman discloses a bean thread that extracts values from an object using read methods for output names, Seidman does not teach or suggest a start position field and a length field that are used to locate data values in an information text string

In addition to the reasons established in sections I and II above, for at least the reasons established in section III above, Applicants respectfully submit that all of the limitations of dependent claim 22 are not taught or suggested by Prunty in view of Chan in further view of Seidman and respectfully request allowance of this claim. Applicants respectfully submit that neither Generous, Smith, Haynie, nor Whalen cures the deficiencies of Prunty in view of Chan in further view of Seidman.

Claim 24:

IV. Prunty in view of Chan does not teach or suggest an EjbServlet that stores a remaining decoded EJB method call result in a memory.

Claim 24 recites, “wherein the EjbServlet stores the remaining decoded EJB method call results in memory.”

The Office Action relied on paragraph 0149 of Haynie to read on the limitations of Claim 24. (Page 13) Paragraph 0149 of Haynie discloses:

The Middleware Proxy App is a stand-alone application that listens for messages from the client, makes the requested EJB call for the client, and then returns the results. The middleware proxy app is multi-threaded and employs bean caching for all stateless session beans. The application also supports stateful session beans through the object ID parameter ("oid"). If the client desires to create a stateful session bean, the client can do so by sending a "CREATE_OBJECT" message to the middleware proxy app. This can be accomplished by using a remote caller to create a remote object (i.e., RemoteCaller.createRemoteObject()). The middleware creates the object and returns an object id to the client.

Haynie discloses an application that makes a requested EJB call for a client, employs bean caching for all stateless session beans, and returns the results. However, Haynie does not teach or suggest an EjbServlet that stores a remaining decoded EJB method call result in a memory. Haynie teaches bean caching for a stateless session bean, not storing the result from decoding an EJB method call into a HTTP response.

In addition to the reasons established in sections I and II above, for at least the reasons established in section IV above, Applicants respectfully submit that all of the limitations of dependent claim 24 are not taught or suggested by Prunty in view of Chan, Seidman, Smith, and in further view of Haynie and respectfully request allowance of this claim. Applicants respectfully submit that neither Generous nor Whalen cures the deficiencies of Prunty in view of Chan, Seidman, Smith, and in further view of Haynie.

Claim 32:

Claim 32 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan.

Independent claim 32 provides:

32. A computing system for accessing an EJB by a non-Java application comprising:

a) a computer storage medium encoded with instructions that when executed by a processor provides: **a non-Java application in communication with a client library, wherein the client library is a linkable library that is dynamically linked to the non-Java application, wherein the non-Java application makes a call to the client library to establish communication between the non-Java application and the client library, wherein the client library comprises a function to take input parameter information from the call, embed the information into an HTTP request, and transfer the HTTP request, and wherein numeric primitive data types of the non-Java application in the input parameter information are converted into a corresponding text representation in the HTTP request;**

b) a computer storage medium encoded with instructions that when executed by a processor provides: an EjbServlet configured to **receive the HTTP request from the client library** and invoke a corresponding method on an EJB; and

c) a computer storage medium encoded with instructions that when executed by a processor provides: a remote method interface (RMI) for invoking methods and returning Java objects between the EjbServlet and the EJB.

Applicants respectfully submit that claim 32 is not taught or suggested by Prunty in view of Chan because they fail to teach or suggest all of the limitations recited in amended claim 32. Specifically, Prunty in view of Chan fails to disclose converting numeric primitive data types of a non-Java application in input parameter information into a corresponding text representation in a HTTP request, or a client library.

Independent claim 32 includes limitations substantially similar to the limitations discussed in sections I and II above. For at least the reasons established above in sections I and II, Applicants respectfully submit that all of the limitations of amended independent claim 32 are not taught or suggested by Prunty in view of Chan and respectfully request allowance of this claim.

Claims Depending From Claim 32:

Claims 33-37 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Prunty in view of Chan.

Dependent claims 33-37 depend directly or indirectly from amended independent claim 32 and incorporate all of the limitations thereof. Accordingly, for at least the reasons established in sections I and II above, Applicants respectfully submit that all of the limitations of dependent claims 33-37 are not taught or suggested by Prunty in view of Chan and respectfully request allowance of this claim.

CONCLUSION

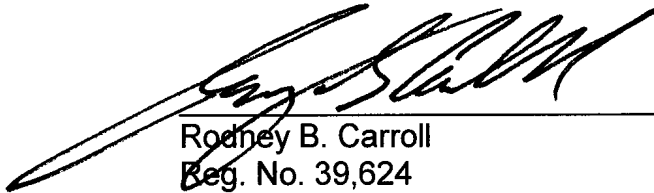
Applicants respectfully submit that the present application is in condition for allowance for the reasons stated above. If the Examiner has any questions or comments or otherwise feels it would be helpful in expediting the application, he is encourage to telephone the undersigned at (972) 731-2288.

The Commissioner is hereby authorized to charge payment of any further fees associated with any of the foregoing papers submitted herewith, or to credit any overpayment thereof, to Deposit Account No. 21-0765, Sprint.

Respectfully submitted,

Date: 3-23-09

CONLEY ROSE, P.C.
5601 Granite Parkway, Suite 750
Plano, Texas 75024
(972) 731-2288
(972) 731-2289 (facsimile)


Rodney B. Carroll
Reg. No. 39,624

ATTORNEY FOR APPLICANTS